Bài 20: Cài đặt thư viện cần thiết.

1. Các thư viện cần thiết:

* Client: webdriverio: Thư viện giúp viết các câu lệnh automation test giúp điều khiển browser.

Trang chủ: https://webdriver.io/

* Test Framework: Mocha: wdio-mocha-framework

Đầu tiên cần tạo ra một project mới:

**Gõ lệnh: npm init**

Điều thông tin cần thiết.

Cài đặt thư viện webdriverio

**npm install –save-dev webdriverio**

(--save: Lưu thông tin vào file config, -dev: chỉ cài vào trong project hiện tại chứ không phải cài trên toàn bộ máy, tránh ảnh hưởng đến các project khác).

Cài đặt thư viện mocha

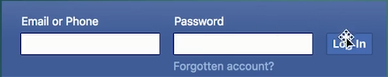
Npm install –save-dev wdio-mocha-framework

Bài 21: Căn bản về các loại Selector

-Mỗi đối tượng Web Element sẽ có một tham chiếu nhất định nào đó trên 1 trang HTML

- Để tương tác được với Web Element chúng ta phải xác định đượv vị trí để tương tác bằng các API tương ứng.

- Vị trí của các Web Element là: Selector value



Ví dụ với phần đăng nhập của facebook thì chúng ta cần xác định các đối tượng username textbox, password textbox và login button. Thì ta cần xác định được để gán giá trị cho các textbox và vị trí button đế ấn nút.

Các loại Selector:

Với WebdriverIO chúng ta có 2 loại :

Css Selector

Xpath

Css selector:

Nếu Web Element là id thì css selector là: #id\_value

Nếu Web Element là class thì css selector là: .class\_value

NẾu là một Link text thì css selector là: =link\_text\_value

Nếu là các Attribue khác, css selector là: [attribute\_name=”value”]

Tham khảo them tại các trang sau: <https://webdriver.io/docs/selectors.html>

Ví dụ:

Css Query Selector:

**const** elem = $('h2.subheading a');

elem.click();

Link Text:

Để lấy được phần tử neo với một nội dung cụ thể, truy vấn sẽ bắt đầu với dấu bằng (=):

<a href="https://webdriver.io">WebdriverIO</a>

Bạn có thể truy vấn bằng cách gọi:

**const** link = $('=WebdriverIO');

console.log(link.getText()); *// outputs: "WebdriverIO"*

console.log(link.getAttribute('href')); *// outputs: "https://webdriver.io"*

Để lấy một phần nội dung văn bản thì ta sẽ xử dụng: \*= ở phía trí chuỗi truy vấn

<a href="https://webdriver.io">WebdriverIO</a>

Ta sẽ truy vấn gọi như sau:

**const** link = $('\*=driver');

console.log(link.getText()); *// outputs: "WebdriverIO*

Phần tử với nội dung text nhất định :

<h1 alt="welcome-to-my-page">Welcome to my Page</h1>

Ta sẽ truy vấn xử lí như sau:

**const** header = $('h1=Welcome to my Page');

console.log(header.getText()); *// outputs: "Welcome to my Page"*

console.log(header.getTagName()); *// outputs: "h1"*

hoặc có thể sử dụng partial text

**const** header = $('h1\*=Welcome');

console.log(header.getText()); *// outputs: "Welcome to my Page"*

Truy vấn qua id và class

<i class="someElem" id="elem">WebdriverIO is the best</i>

Ta sẽ truy vấn bằng cách gọi:

**const** classNameAndText = $('.someElem=WebdriverIO is the best');

console.log(classNameAndText.getText()); *// outputs: "WebdriverIO is the best"*

**const** idAndText = $('#elem=WebdriverIO is the best');

console.log(idAndText.getText()); *// outputs: "WebdriverIO is the best"*

**const** classNameAndPartialText = $('.someElem\*=WebdriverIO');

console.log(classNameAndPartialText.getText()); *// outputs: "WebdriverIO is the best"*

**const** idAndPartialText = $('#elem\*=WebdriverIO');

console.log(idAndPartialText.getText()); *// outputs: "WebdriverIO is the best"*

Truy vấn qua Tag Name:

<my-element>WebdriverIO is the best</my-element>

Ta sẽ thực hiện gọi như sau:

**const** classNameAndText = $('<my-element />');

console.log(classNameAndText.getText()); *// outputs: "WebdriverIO is the best"*

Truy cập qua xPath:

Truy cập lần lượt thông qua các thẻ từ ngoài vào trong:

Có 2 loại là Absolute xPath: Viết lần lượt từ thẻ ngoài cùng vào trong. Thường không ai dùng. Do ta có thể thay đổi 1 thẻ dẫn đến bị sai.

<html>

<body>

<p>foobar</p>

<p>barfoo</p>

</body>

</html>

Ta sẽ truy cập bằng cách gọi:

**const** paragraph = $('//BODY/P[1]');

console.log(paragraph.getText()); *// outputs: "barfoo"*

**const** parent = paragraph.$('..');

console.log(parent.getTagName()); *// outputs: "body"*

Relative xPath:

Cú pháp:

//tagname[@attributeName=’attributeValue’]

//\*[@attributeName=’attributeValue’]

Ví dụ:

//input[@type=’text’]

//label[@id=’id\_name’]

//input[@value=’value’]

//a[@href=’https://google.com/’]

//img[@src=’/images/teo.png’]

//\*[@class=’className’]

Bài 22: Các công cụ lập trình hỗ trợ lấy WebElement

* Firebug/ FirePath

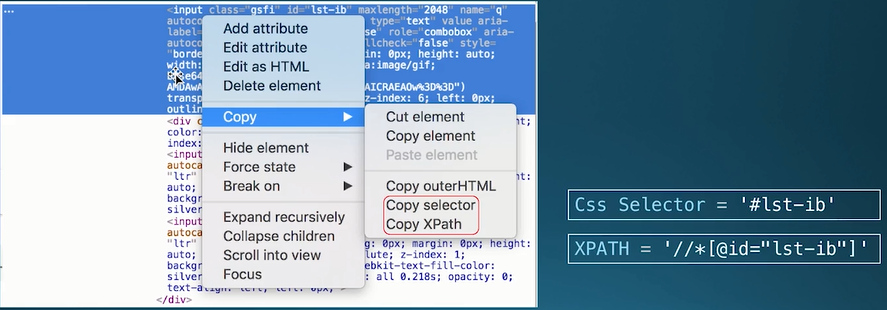
+ 2 extension này là trên Firefox, khá nổi tiếng.

+ Nhưng không theo kịp Firefox version nên hơi bất tiện. Nhưng ngày này không mấy ai dùng do nhiều pluggin hỗ trợ

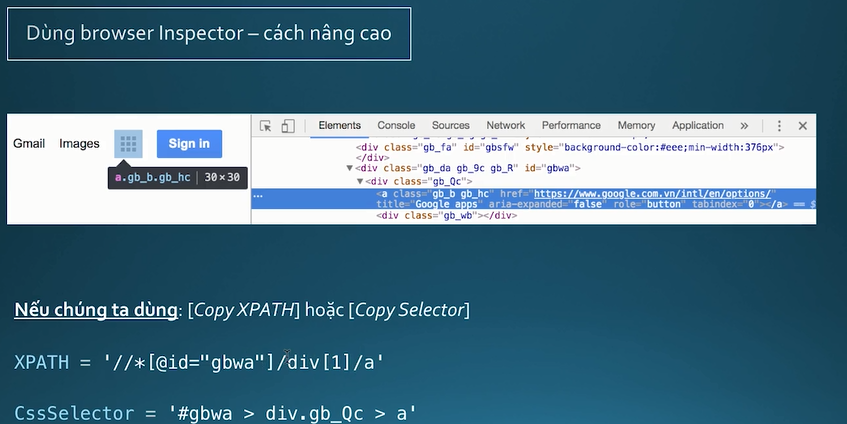
Inspector on Brower:

Click chuột phải cần đối tượng cần xem -> Inspect

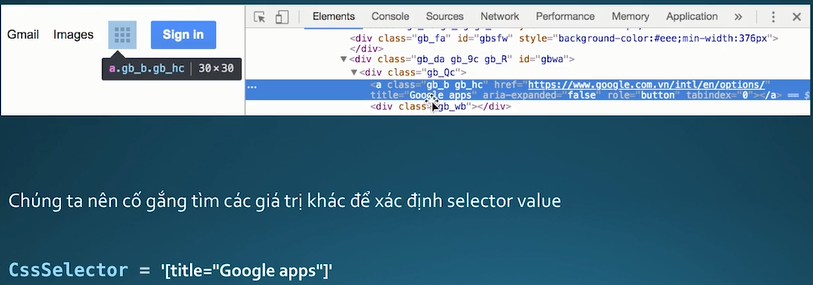
Sau khi tìm được đoạn html của Element cần tìm. Chuột phải chọn:

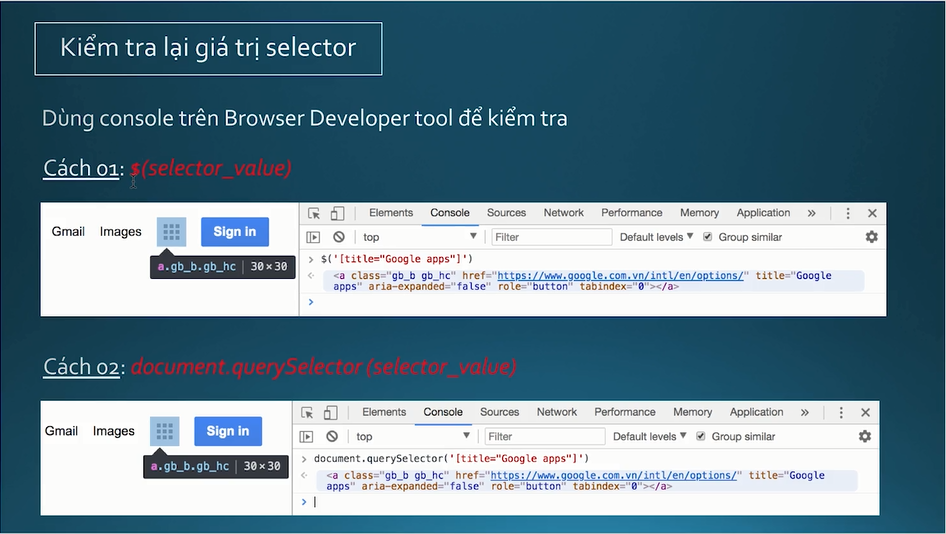


Tuy nhiên trong một số trường hợp, nội dung selector hay xpath lấy được rất khó đọc



Để giải quyết được vấn đề này thì ta cần phải thực hiện lấy ra selector khác dễ dàng đọc hơn.





Bài 24: Selenium Server và Browser Drivers.

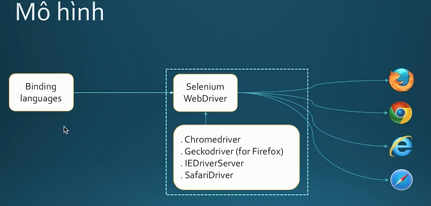
1. Selenium standalone server

Điều khiển các browser thông qua các browser driver (ChormeDriver, geckodriver – Firefox, safari Driver…)

1. Chúng ta chỉ thật sự cần Selenim server cho những mục đích.

Chúng ta muốn dùng hệ thống selenium grid để phân bổ test trên nhiều môi trường khác nhau.

Dùng để yêu cầu resource trên 1 remote machine nới chúng ta cần 1 tài nguyên browser nào đó.



Binding langugues: Là ngôn ngữ được viết chuyển sang nodejs, java, c#... Lúc đó viết lại api ở ngôn ngữ đó. Sau khi viết xong sẽ gửi sang cho selenium server. Selenium server sẽ chọn các driver tương ứng để chuyển sang test trên các browser.

Chormedriver:

Dùng để điều khiển ChormeBrowser

Chormedriver được thiết kế như 1 stand-alone server: ví dụ khi khởi động chorme server có port 9515, ở client thay vì trỏ tới selenium webdriver ta trỏ tới host(ip của máy đăng chạy chormedriver) và port trên là ta có thể giao tiếp client <-> server

Bài 25: Khởi động Chorrme Driver

Trước khi chạy automation thì ở client viết script và gửi tới server(chorme drivers) để điều khiển thực thi của mình.

* Có 2 cách:

Cách 1:

+ Download và khởi động trực tiếp Chorme driver: https://sites.google.com/a/chromium.org/chromedriver/

+ Unzip và double click để chạy.

+ Ưu điểm là chúng ta có thể khởi động trên một máy nào đó và yêu cầu browser test trên máy đó.

+ Nhược điểm là chúng ta phải tự khởi động.

Cách 2: wdi-selenium-standalone-service

DÙng để download và khởi động ccs browser version một các tự khởi động.

npm install –save-dev wdio-selenium-standalone-service

Ưu điểm là hcungs ta có thể không cần tự khởi động browser driver bằng tay nữa.

Nhược điểm là không cần resource browser test trên máy khác được .

Bài 26: File cấu hình cho việc thực thi test và các thành phần quan trọng

Vào trang chủ webdriver.io phần Guide/Testuner/Config để xem file config

Khi test cần

Spects: Đưa vào các file test script cần test

Host: Địa chỉ IP của máy để yêu cầu resource test

Port: Là port của selenium server hoặc browser driver

Path: Chỉ cần khi ta tự khởi tạo browser driver.

exports.config ={

specs: [],

host:'localhost',

port: 4444,

path: '/',

capabilities: [{

browserName: 'chrome'

}],

framework: 'mocha',

mochaOpts: {

ui: 'bdd',

timeout:600000

},

logLevel: 'info'

}

**CHROME DRIVER NPM**

Nếu trong trường hợp chạy Chorme driver bị lỗi do cấu hình không chạy đươc thì ta sẽ thực hiện dùng Chormedriver qua npm để làm server như sau:

Gõ: npm install chromedriver

Kiểm tra xem cài đặt được chưa: chromedriver -v

Nếu ra phiên bản thì có nghĩa đã cài thành công.

Cấu hình port và path:

**Chromedriver –port=4444 –url-base=/wd/hub –verbose**

**Ngoài ra nếu bị lỗi thì trog phần package.json ta cấu hình các phiên bản về cùng 1 phiên bản rồi install lại**

Bài 27: Cấu hình Autocompletion để viết code dễ dàng hơn.

Để có gợi ý khi ta viết code với webdriverio, chúng ta nên cấu hình code autocompletions:

Npm install –save-dev @types/node

Npm install –save-dev @types/webdriverio

Bài 28: URL

Mở một trang web lên:

browser.url(“đường dẫn”);

ví dụ:

browser.url('https://webdriver.io');

Lấy địa chỉ hiện tại của browser:

browser.getUrl()

Bài 29: API Điền giá trị vào field

API: setValue

Cú pháp: browser.setValue(selector, values)

Dấu $ đại diện cho browser.element

Dấu $$ tìm tập hợp một đối tượng.

Để lấy value: browser.getValue(selector)

Để ngắn gọn thì ta thường tạo ra một biến đại diện cho một selector:

**const** input = $('.input');

Ví dụ: Mở trang google.com sau đó thực hiện thêm dòng chữ vào mục search: automation test

describe('API command', function(){

it('Open a page', function(){

browser.url("https://www.google.com");

browser.pause(2000);

const input= $('.gLFyf')

input.setValue('Automation Test');

console.log("111111111111111:", input.getValue());

browser.pause(2000);

})

})

Bài 30: API chờ đối tượng hiển thị

API waitForDisplayed waitFormVisible

Chờ một element nào hiển thị lên trang web trước khi tương tác tại khi vừa tạo nếu chưa có elemenet sẽ xảy ra lỗi.

Cú pháp:

browser.waitForDisplayed(ms, reverse, error)

ms: thời gian hiển thị

reverse: Nếu true nó sẽ đợi ngược lại(Mặc định là false)

error:

Với api này nó sẽ 500mls sẽ thực hiện chờ 1 lần. Chờ đến khi nào hết chỗ ms truyền vào mà không lấy được thì sẽ báo lỗi.

describe('API command', function(){

it('Open a page', function(){

browser.url("https://www.google.com");

const elem = $('.gLFyf')

elem.waitForDisplayed(3000);

$('.gLFyf').setValue('Automation Test');

})

})

Bài 31: Click lên một đối tượng

API click

Trang web từ nay sẽ dùng để test

http://the-internet.herokuapp.com/

<button id="myButton" onclick="document.getElementById('someText').innerHTML='I was clicked'">Click me</button>

<div id="someText">I was not clicked</div>

it('should demonstrate the click command', () => {

**const** myButton = $('#myButton')

myButton.click()

**const** myText = $('#someText')

**const** text = myText.getText();

assert(text === 'I was clicked'); *// true*

})

it('should fetch menu links and visit each page', () => {

**const** links = $$('#menu a');

links.forEach((link) => {

link.click();

});

});

describe('API command', function(){

it('Open a page', function(){

browser.url('http://the-internet.herokuapp.com/login');

var username= $('#username');

var password=$('#password');

var button= $('.radius');

//var button= $('[type="submit"]');

username.waitForDisplayed(1000);

username.setValue('tomsmith');

password.waitForDisplayed(1000);

password.setValue('SuperSecretPassword!');

button.waitForDisplayed(1000);

button.click();

browser.pause(2000);

})

})

Bài 33: xử lí dropdown menu

API selectByAttribute, selectByIndex, selectByVisibleText

selectByAttribte

Cú pháp: browser.selectByAttribute(attribute, value)

<select id="selectbox">

<option value="someValue0">uno</option>

<option value="someValue1">dos</option>

<option value="someValue2">tres</option>

<option value="someValue3">cuatro</option>

<option value="someValue4">cinco</option>

<option name="someName5" value="someValue5">seis</option>

</select>

it('Should demonstrate the selectByAttribute command', () => {

**const** selectBox = $('#selectbox');

**const** value = selectBox.getValue();

console.log(value); *// returns "someValue0"*

selectBox.selectByAttribute('value', 'someValue3');

console.log(selectBox.getValue()); *// returns "someValue3"*

selectBox.selectByAttribute('name', 'someName5');

console.log(selectBox.getValue()); *// returns "someValue5"*

});

Select by index

<select id="selectbox">

<option value="someValue0">uno</option>

<option value="someValue1">dos</option>

<option value="someValue2">tres</option>

</select>

it('Should demonstrate the selectByIndex command', () => {

**const** selectBox = $('#selectbox');

console.log(selectBox.getValue()); *// returns "someValue0"*

selectBox.selectByIndex(1);

console.log(selectBox.getValue()); *// returns "someValue1"*

});

selectByVisibleText

it('demonstrate the selectByVisibleText command', () => {

**const** selectBox = $('#selectbox');

console.log(selectBox.getText('option:checked')); *// returns "uno"*

selectBox.selectByVisibleText('cuatro');

console.log(selectBox.getText('option:checked')); *// returns "cuatro"*

})

Bài 34: API Xử lí alert

it('Alert Accept', function(){

browser.url('http://the-internet.herokuapp.com/javascript\_alerts');

var buttonConfirm= $('[onclick="jsConfirm()"]');

buttonConfirm.waitForDisplayed(3000);

buttonConfirm.click();

browser.pause(3000);

browser.acceptAlert();

browser.pause(3000);

})

Sự kiện khi ấn vào nút cancel trên alert

it('Alert Accept', function(){

browser.url('http://the-internet.herokuapp.com/javascript\_alerts');

var buttonConfirm= $('[onclick="jsConfirm()"]');

buttonConfirm.waitForDisplayed(3000);

buttonConfirm.click();

browser.pause(3000);

browser.dismissAlert();

browser.pause(3000);

})

Đối với sự kiện alert chỉ có nút ok ta cũng vẫn sử dụng: button.acceptAlert();

it('Alert Accept', function(){

browser.url('http://the-internet.herokuapp.com/javascript\_alerts');

const button = $('[onclick="jsAlert()"]')

button.waitForDisplayed(3000);

button.click();

button.acceptAlert();

browser.pause(3000);

})

Nhập dữ liệu vào trong Promt alert:

it('Alert Accept', function(){

browser.url('http://the-internet.herokuapp.com/javascript\_alerts');

const button = $('[onclick="jsPrompt()"]')

button.waitForDisplayed(3000);

button.click();

button.sendAlertText("abc");

button.acceptAlert();

browser.pause(3000);

})

Bài 37: Lấy Attribute

describe('API command', function(){

it('Alert Accept', function(){

browser.url('http://the-internet.herokuapp.com/login');

const form = $('form')

const attr = form.getAttribute('method')

console.log("!!!!!!!!!!!!!!!!!!!",attr) // outputs: "post"

})

})

Bài 39: Checkbox

it('upload file', function(){

browser.url('http://the-internet.herokuapp.com/checkboxes');

var check= $$('[type="checkbox"]');

browser.pause(3000);

check[0].click();

browser.pause(3000);

})

Bài 44: Page Object

Là một design patent

Là cách chúng ta ánh xạ một page như 1 đối tượng. Element trên page là các thuộc tính và các method tác động lên các element đó là các phương thức.



Tầm quan trong của PageObject



Bài 45. Triển khia pageObject loại 1.

Tạo một file js có nội dung như sau, export ra một instance

const TXT\_USERNAME= '#username';

const TXT\_PASSWORD='#password';

const BTN\_LOGIN='[type="submit"]';

class LoginPage{

get userName(){

return $(TXT\_USERNAME);

}

get password(){

return $(TXT\_PASSWORD);

}

get loginBtn(){

return $(BTN\_LOGIN);

}

}

module.exports = new LoginPage();

Ta gọi sử dụng lại như sau:

var LoginPage= require('../pageObjects/LoginPage');

describe('API command', function(){

it('upload file', function(){

browser.url('http://the-internet.herokuapp.com/login');

LoginPage.userName.setValue('tomsmith');

LoginPage.password.setValue('SuperSecretPassword!');

browser.pause(3000);

LoginPage.loginBtn.click();

browser.pause(5000);

})

})

Bài 47: Page Object loại 2

Đối với một số page thì sẽ lặp đi lặp lại một công việc nhập dữ liệu và ấn click button. Ví dụ như trong trang login thì lặp lại công việc nhập username và password sau đó nhấn nút submit để login. Thì với nhiều test ta thực hiện đi thực hiện lại công việc này. Thì thay vào đó ta sẽ tạo ra một phương thức thực hiện luôn 3 công việc này như vậy trong form thì ta chỉ cần gọi lại phương thức đó thôi.

Ta tạo ra một file lưu class thực hiện page object có nội dung như sau:

const TXT\_USERNAME= '#username';

const TXT\_PASSWORD='#password';

const BTN\_LOGIN='[type="submit"]';

class LoginPage{

get userName(){ return $(TXT\_USERNAME); }

get password(){ return $(TXT\_PASSWORD); }

get loginBtn(){ return $(BTN\_LOGIN); }

LoginToPage(username, password){

this.userName.setValue(username);

this.password.setValue(password);

this.loginBtn.click();

}

}

module.exports = new LoginPage();

Để thực hiện thì ta sẽ thực hiện như sau:

var LoginPage = require('../pageObjects/LoginPage');

describe('API command', function () {

it('upload file', function () {

browser.url('http://the-internet.herokuapp.com/login');

LoginPage.LoginToPage('tomsmith','SuperSecretPassword!');

browser.pause(3000);

})

})

Bài 49. Triển khai Page Object loại 3

Thường được dùng cho loại project phức tạp

Khi tương tác trên một pageobject người ta chỉ càn phương thức để tương tác trên webelement của một page. Ví dụ trang login thì có các phương thức: InputUsername, InputPassword, InputButton

const TXT\_USERNAME= '#username';

const TXT\_PASSWORD='#password';

const BTN\_LOGIN='[type="submit"]';

class LoginPage{

inputUserName(name){

$(TXT\_USERNAME).setValue(name);

}

inputPassword(password){

$(TXT\_PASSWORD).setValue(password);

}

clickOnLoginBtn(){

$(BTN\_LOGIN).click();

}

}

module.exports = new LoginPage();

var LoginPage = require('../pageObjects/LoginPage');

describe('API command', function () {

it('upload file', function () {

browser.url('http://the-internet.herokuapp.com/login');

LoginPage.inputPassword('SuperSecretPassword!');

LoginPage.inputUserName('tomsmith');

LoginPage.clickOnLoginBtn();

browser.pause(3000);

})

})

Nhược điểm:

Ví dụ trong testcase cần check size button hay màu sắc thì phải thêm các phương thức trả lại thông tin đó. Nhưng rất ít page trả lại như vậy .

Bài 50: Kỹ thuật điều hướng các pageobject bằng action chain

Giúp viết testscript và gọi ra các action method gọn gàng hơn.

Với những thao tác không làm chuyển trang thì sẽ trả lại chỉnh trang đó(return this)

Nếu thao tác nào làm chuyển trang thì ta sẽ return trang đích.

Ví dụ với Page login khi ta nhập username, password thì sẽ không làm chuyển trang. Nhưng khi ta click vào nút login thì nó sẽ chuyển sang trang mới.

Ta sẽ sử dụng pageobject loại 3.

Ta tạo ra page object Login

Một page object SecuPage.

Nội dung như sau:

var LoginPage = require('../pageObjects/LoginPage');

describe('API command', function () {

it('upload file', function () {

browser.url('http://the-internet.herokuapp.com/login');

LoginPage

.inputPassword('SuperSecretPassword!')

.inputUserName('tomsmith')

.clickOnLoginBtn()

.getSecureAreaText();

browser.pause(3000);

})

})

const TXT\_USERNAME= '#username';

const TXT\_PASSWORD='#password';

const BTN\_LOGIN='[type="submit"]';

var SecuPage= require('./SecuPage');

class LoginPage{

inputUserName(name){

$(TXT\_USERNAME).setValue(name);

return this;

}

inputPassword(password){

$(TXT\_PASSWORD).setValue(password);

return this;

}

clickOnLoginBtn(){

$(BTN\_LOGIN).click();

return SecuPage;

}

}

module.exports = new LoginPage();

const LOGIN\_OK\_CONTENT='h2';

class SecuPage{

getSecureAreaText(){

let loginOkContent= $(LOGIN\_OK\_CONTENT).getValue();

console.log("!!!!!!!!!!!!:",loginOkContent);

}

}

module.exports = new SecuPage();

Bafi 52: Kế thừa trong pageObject

Giúp kế thừa lại một số những phương thức tránh lặp đi lặp lại nhiều lần ví dụ như trong tất cả các page đều có mở một trang web. Ví dụ ta muốn page nào cũng có một phương thức mở trang web đó được kế thừa thì ta sẽ làm như sau:

Tạo ra một page Base.js có nội dung như sau:

class Base{

openPage(url){

browser.url(url);

return this;

}

}

module.exports= Base;

Ở các class page khác thì ta sẽ kế thừa lại class Base

var Base= require('./Base');

class LoginPage extends Base{

Ta sẽ sử dụng lại ở các trang:

LoginPage.openPage('http://the-internet.herokuapp.com/login');

LoginPage

.inputPassword('SuperSecretPassword!')

.inputUserName('tomsmith')

.clickOnLoginBtn()

.getSecureAreaText();

browser.pause(3000);

Để . liên tục được thì cần nhớ là return this ở trong phương thức openPage()

Bafi 54: Before suite và aftersuite

Before chạy một lần và chạy đầu tiên. Thường dùng để khởi tạo một dữ liệu gì đó.

Aftersuite dùng để xóa dữ liệu sau cùng.

before('Data prepareation', ()=>{

console.log("2222222222222222222222222Prepare data");

LoginPage.openPage('http://the-internet.herokuapp.com/login');

});

after('clear data', ()=>{

console.log("Destructor data");

})

Bài 55: beforeEach và afterEach

BeforeEach chạy trước một test case it 1 lần.

afterEach chạy sau một it một lần.

beforeEach(‘Refresh page’, function(){

}

beforeEach(‘After page’, function(){

}

Bài 38: Lấy size của một element web